Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/US04/032352

International filing date: 30 September 2004 (30.09.2004)

Document type: Certified copy of priority document

Document details: Country/Office: US
Number: 60/576,643
Filing date: 01 June 2004 (01.06.2004)

Date of receipt at the International Bureau: 19 November 2004 (19.11.2004)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)

World Intellectual Property Organization (WIPO) - Geneva, Switzerland
Organisation Mondiale de la Propriété Intellectuelle (OMPI) - Genève, Suisse

1247545

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

November 09, 2004

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE.

APPLICATION NUMBER: *60/576,643*
FILING DATE: *June 01, 2004*
RELATED PCT APPLICATION NUMBER: *PCT/US04/32352*

Certified by

Jon W Dudas

Acting Under Secretary of Commerce
for Intellectual Property
and Acting Director of the U.S.
Patent and Trademark Office

PTO/SB/16(01-04)
Approved for use through 07/31/2006. OMB 0651-0032
Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c).

EV 378674198 US
Express Mail Label No.

16367 U.S. PTO 060104

22151 U.S. PTO 60/576643 060104

INVENTOR(s)

Given Name (first and middle [if any])	Family Name or Surname	Residence (City and either State or Foreign Country)
Nikil	Dutt	Irvine, California
Mohammad H.	Reshadi	Irvine, California

☐ *Additional Inventors are being name on the ___ separately numbered sheets attached hereto*

TITLE OF THE INVENTION (280 characters max)

SYSTEMS AND METHODS FOR SIMULATING INSTRUCTION SET ARCHITECTURES

CORRESPONDENCE ADDRESS

DIRECT ALL CORRESPONDENCE TO:

☒ Customer Number **34313**

OR *TYPE CUSTOMER NUMBER HERE*

☒ Firm or Individual Name	**Orrick, Herrington & Sutcliffe LLP**				
Address	**4 Park Plaza, Suite 1600**				
City	**Irvine**	State	**California**	ZIP	**92614-2558**
Country	**US**	Telephone	**949-567-6700**	Fax	**949-567-6710**

ENCLOSED APPLICATION PARTS (*check all that apply*)

☒	Specification	*Number of pages*	55	☐	CD(s) Number
☒	Drawing(s)	*Number of sheets*	3	☒	Other (specify) return postcard
☐	Application Data Sheet. See 37 CFR 1.76				

METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONS APPLICATION FOR PATENT

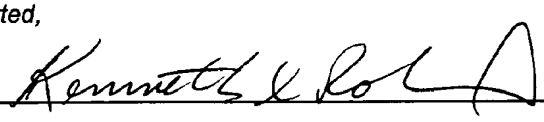
		FILING FEE AMOUNT($)
☒	Applicant claims small entity status. See 37 CFR 1.27.	
☐	A check or money order is enclosed to cover the filing fees	
☒	The commissioner is hereby authorized to charge filing fees and credit Deposit Account Number: 15-0665	**$80.00**
☐	Payment by credit card. Form PTO-2038 is attached.	

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

☐ No.

☒ Yes, the name of the U.S. Government agency and the Government contract number are: NSF Contract CCT-0203813

Respectfully submitted,

SIGNATURE  Date: 6/1/2004

TYPED or PRINTED NAME Kenneth S. Roberts REGISTRATION NO. 38,283 *(if appropriate)*

TELEPHONE 949-567-6700 DOCKET NUMBER: 703538.4050

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

This collection of information in required by 37 CFR 1.51. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 USC 122 and 37 CFR 1.14. This collection is estimated to take 8 hours to complete, including gathering, preparing and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, US Patent and Trademark Office, US Department of Commerce, P.S. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: **Mail Stop Provisional Application, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

UNITED STATES PATENT APPLICATION
FOR

SYSTEMS AND METHODS FOR SIMULATING INSTRUCTION SET ARCHITECTURES

INVENTOR:

NIKIL DUTT
MOHAMMAD H. RESHADI

PREPARED BY:

ORRICK, HERRINGTON & SUTCLIFFE LLP

FOUR PARK PLAZA, SUITE 1600

IRVINE, CA 92614-2558

SYSTEMS AND METHODS FOR SIMULATING INSTRUCTION SET ARCHITECTURES

FIELD OF THE INVENTION

[001] The invention relates generally to instruction set architecture simulation and more particularly, to systems and methods for the efficient modeling of processors and automatic generation of retargetable instruction set simulators.

BACKGROUND INFORMATION

[002] An instruction-set architecture (ISA) simulators are tools that run on a host machine to mimic the behavior of running an application program on a target machine. Instruction-set simulators are valuable tools in the development of new programmable architectures. They are used to validate architecture and compiler designs, as well as to evaluate architectural design decisions during design space exploration. **FIG. 1** depicts a traditional interpretive simulation technique 10 that is flexible but slow. In technique 10, an instruction 11 stored in program memory 12 is fetched at 13, decoded at 14, and executed at 15 during run time 16. Instruction decoding is a time consuming process in a software simulation.

[003] **FIG. 2** depicts a conventional compiled simulation technique 18 that performs compile time decoding of an application program 20 to improve the simulation performance. Specifically, the application program 20 is compiled in a simulation compiler 21 to create a decoded program 22. That decoded program 22 is passed through a code generation process 23 to create a host assembly 24 stored in program memory 12, which is then executed at 15 by the host 25.

[004] To improve the simulation speed further, static compilation based techniques move the instruction scheduling into the compilation phase. However, all compiled simulators rely on the assumption that the complete program code is known before the simulation starts and is further more run-time static. Due to this assumption many application domains are excluded from the utilization of compiled simulators. For example, embedded systems that use external program memories cannot use compiled simulators since the program code is not predictable prior to runtime. Similarly, compiled simulators are not applicable in embedded systems that use processors having multiple instruction sets. These processors can switch to a different instruction set mode at run time. For instance, the ARM processor uses the Thumb (reduced bit-width) instruction set to reduce power and memory consumption. This dynamic switching of instruction set modes cannot be considered by a simulation compiler 21, since the selection depends on run-time values and is not predictable. Furthermore, applications with run-time dynamic program code, as provided by operating systems (OS), can not be addressed by compiled simulators.

[005] Also, in the past years, performance has been the most important quality measure for the ISA simulators. However, more recently, retargetability has become an important concern, particularly in the area of the embedded systems and system-on-chip (SoC) design. A retargetable ISA simulator requires a generic model, supported by a language, to describe the architecture and its instruction set. The simulator uses the architecture description to decode instructions of the input program and execute them. The challenge is to have a model that is efficient in terms of both quality of the description and performance of the simulator. To have a high quality description, the model must easily capture the architectural information in a natural, compact and manageable form for a wide range of architectures. On the other hand, to generate a

high performance simulator and to reduce the operations that the simulator must do dynamically at run time, the model should provide as much static information as possible about the architecture and its instruction set.

[006] Designing an efficient model that captures a wide range of architectures is difficult because such architectures have different instruction-set format complexities. There is a tradeoff between speed and retargetability in ISA simulators. Some of the retargetable simulators use a very general processor model and support a wide range of architectures but are slow, while others use some architectural or domain specific performance improvements but support only a limited range of processors. Also in some description languages, deriving a fast simulator requires lengthy descriptions of all possible formats of instructions.

[007] Accordingly, there is a need for improved ISA simulators and simulation methods that address the above concerns and provide advantages over conventional systems and methods, such as by providing the efficient modeling of processors and automatic generation of retargetable instruction set simulators.

BRIEF DESCRIPTION OF THE FIGURES

[008] The details of the invention, including fabrication, structure and operation, may be gleaned in part by study of the accompanying figures, in which like reference numerals refer to like segments.

[009] **FIG. 1** depicts a flow diagram of a prior art interpretive simulation.

[010] **FIG. 2** depicts a flow diagram of a prior art compiled simulation.

[011] **FIG. 3** depicts a flow diagram of one exemplary embodiment of a instruction set compiled simulation method.

BRIEF DESCRIPTION

[012] The systems and methods described herein provide improved instruction set architecture (ISA) simulators and methods of generating the same. More specifically, the systems and methods provide an instruction set compiled simulation (IS-CS) method that uses a templatized approach to generate fast ISA simulators that combine the benefits of both compiled and interpretive simulation. Also provided is a hybrid compiled simulation method for reducing compilation time in the generation of ISA simulators. The improved ISA simulators discussed herein can also be efficiently retargeted to various processor architectures. The various embodiments of these systems and methods can be implemented with the IS-CS method or with other simulation methods.

[013] **FIG. 3** depicts a flow diagram of IS-CS method 100. As stated above, the IS-CS method 100 can be used for generation of fast instruction-set simulators by combing the performance of traditional compiled simulation with the flexibility of interpretive simulation. The IS-CS method 100 achieves high performance for at least two reasons. First, the time consuming instruction decoding process is moved to compile time while maintaining the flexibility of interpretive simulation by executing interpretively during run-time. The simulator can recognize if an instruction is modified at run-time, in which case the instruction is re-decoded prior to execution. Second, an instruction abstraction method is used to generate aggressively optimized decoded instructions that further improve simulation performance. The IS-CS method 100 delivers better performance than conventional simulation methods that have the flexibility of interpretive simulation.

[014] The application program 102, written in C/C++, is compiled using the GCC compiler 104 configured to generate binary 106 for the target machine. The instruction decoder 108

decodes one binary instruction at a time to generate the decoded program 110 for the input application 102. The decoded program 110 is compiled by C++ compiler 114 and linked with the simulation library (not shown) to generate the simulator 134. The simulator 134 recognizes if the previously decoded instruction has changed and initiates re-decoding 126 of the modified instruction. If any instruction is modified during execution 132 and subsequently re-decoded at 126, the location in instruction memory 120 is updated with the re-decoded instruction 122. To improve the simulation speed an instruction abstraction method 200 is used that generates optimized decoded instructions and, as a result, the computation during run-time is minimized.

[015] In traditional interpretive simulation (e.g., Simplescalar) the decoding and execution of binary instructions are done using a single monolithic function. This function has many if-then-else and switch/case statements that perform certain activities based on bit patterns of opcode, operands, addressing modes etc. In advanced interpretive simulation (e.g., LISA) the binary instruction is decoded and the decoded instruction contains pointers to specific functions. There are many variations of these two methods based on efficiency of decode, complexity of implementation, and performance of execution. However, none of these techniques utilize the fact that a certain class of instructions may have a constant value for a particular field of the instruction. For example, a majority of the ARM instructions execute unconditionally (condition field has value always). It is not time efficient to check the condition for such instructions every time they are executed.

[016] When certain input values are known for a class of instructions, the partial evaluation technique can be applied. This technique is well known to one of skill in the art and is discussed in more detail in Y. Futamura, "Partial Evaluation of Computation Process: an Approach to a Compiler-Compiler." *Systems, Computers, Controls*, Volume 2(5), Pages 45-50, 1971, which is

fully incorporated by reference herein. The effect of partial evaluation is to specialize a program with part of its input to get a faster version of the same program. To take advantage of such situations separate functions are utilized for each and every possible format of instructions so that the function can be optimized by the compiler at compile time and produce the best performance at run time. This can be very difficult in practice.

[017] For example, the ARM data processing instructions can have 16 conditions, 16 operations, an update flag (true/false), and one destination followed by two source operands. The second source operand, called shifter operand, has three fields: operand type (reg/imm), shift options (5 types) and shift value (reg/imm). In total, the ARM data processing instructions have 16 x 16 x 2 x 2 x 5 x 2 = 10240 possible formats.

[018] The instruction abstraction method 200 disclosed herein (see Appendix B.1)defines instruction classes, where each class contains instructions with similar formats. This information can often be readily available from the instruction set architecture manual. For example, six instruction classes for the ARM processor can be defined: Data Processing, Branch, LoadStore, Multiply, Multiple Load-Store, Software Interrupt, and Swap. Next, a set of masks for each instruction class can be defined. The mask preferably includes '0', '1' and 'x' symbols. A '0' ('1') symbol in the mask matches with a '0' ('1') in the binary pattern of the instruction at the same bit position. An 'x' symbol matches with both '0' and '1'. For example, the masks for the data processing instructions are shown below:

"xxxx-001x xxxx-xxxx xxxx-xxxx xxxx-xxxx"

"xxxx-000x xxxx-xxxx xxxx-xxxx xxx0-xxxx"

"xxxx-000x xxxx-xxxx xxxx-xxxx 0xx1-xxxx"

[019] C++ templates can be used to implement the functionality for each class of instructions. For example, the pseudo code for the data processing template is shown below in Template 1. The template has four parameters viz., condition, operation, update flag, and shifter operand. The shifter operand is a template having three parameters viz., operand type, shift options and shift value.

Template for Data Processing Instructions

```
template <class Cond, class Op, class Flag, class SftOper> class DataProcessing :
{
  SftOper _sftOperand;
  Reg _dest, _src1;
public:
.....
  virtual void execute()
  {
    if (Cond::execute())
    {
      _dest = Op::execute(_src1, _sftOperand.getValue());
      if (Flag::execute())
      {
        // Update Flags
        .....
      }
    }
  }
};
```

Template 1

[020] Appendix A, which is incorporated by reference herein, includes thirteen templates used to implement the functionality for each class of instructions in the ARM architecture.

[021] Appendix B, which is also incorporated by reference herein, discusses the IS-CS method 100 in more detail, including the instruction abstraction method 200, the instruction decoder 108 and the simulation engine 134 in more detail.

[022] Also provided herein is a retargetable simulation framework 300 (see Fig. 1; Appendix B.2), which can be used to efficiently retarget an instruction set simulator to a different processor architecture. The retargetable simulation framework 300 includes a generic instruction

model 302 and a generic instruction decoder 304. The retargetable framework 300 can be used with the IS-CS method 100 or with other simulation methods. The retargetable framework 300 is discussed in more detail in Appendix B.

[023] Also provided herein is a hybrid compiled simulation method 400 (See Fig. 3; Appendix B.3). This hybrid simulation method 400 combines both static and dynamic compilation to generate source code of a decoder customized to the input program, as opposed to generating a source code that is equivalent to the input program as done in conventional methods. Like the retargetable framework 300, the hybrid compiled simulation method 400 can be used with the IS-CS method 100 or with other simulation methods. The hybrid compiled simulation method 400 is discussed in more detail in Appendix B.

[024] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. For example, each feature of one embodiment can be mixed and matched with other features shown in other embodiments. Features and processes known to those of ordinary skill may similarly be incorporated as desired. Additionally and obviously, features may be added or subtracted as desired. Accordingly, the invention is not to be restricted except in light of the attached claims and their equivalents.

What is claimed is:

1. A method of simulating an instruction set architecture, comprising:

decoding an input program in an instruction decoder to generate a temporary program, wherein the input program comprises a plurality of instructions being classifiable into one or more instruction classes, and wherein the decoded program is generated for each instruction class;

compiling the temporary program to generate a decoded program with optimized instructions;

storing the decoded program in instruction memory;

generating a simulator configured to simulate an instruction set architecture during a run-time; and

operating the simulator to fetch each optimized instruction from the instruction memory, execute the optimized instruction if it has not been modified during run-time and re-decode the optimized instruction if it has been modified during run-time.

2. The method of claim 1, further comprising updating the optimized decoded instruction in instruction memory with the re-decoded instruction.

3. The method of claim 1, wherein decoding the input program in the instruction decoder further comprises:

selecting a template for each instruction in the input program;

customizing the template for each instruction with a set of parameters; and

instantiating the customized template into the temporary program.

4. The method of claim 3, wherein each instruction in the input program comprises a plurality of fields each having a value.

5. The method of claim 4, wherein customizing the template comprises extracting the value from each field in the instruction and assigning the value to the template for that instruction.

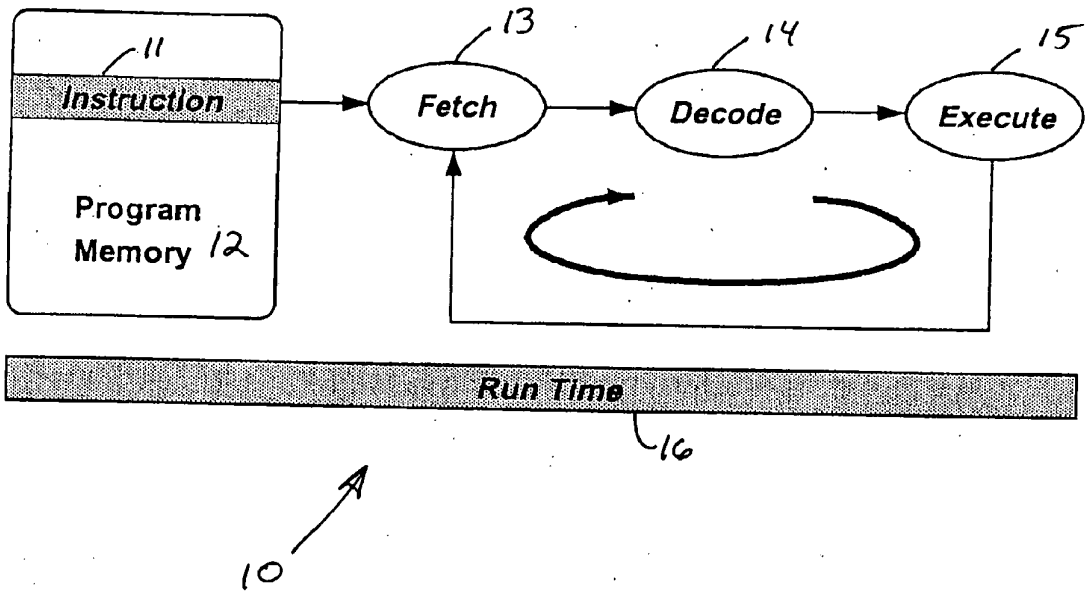


FIG. 1

PRIOR ART

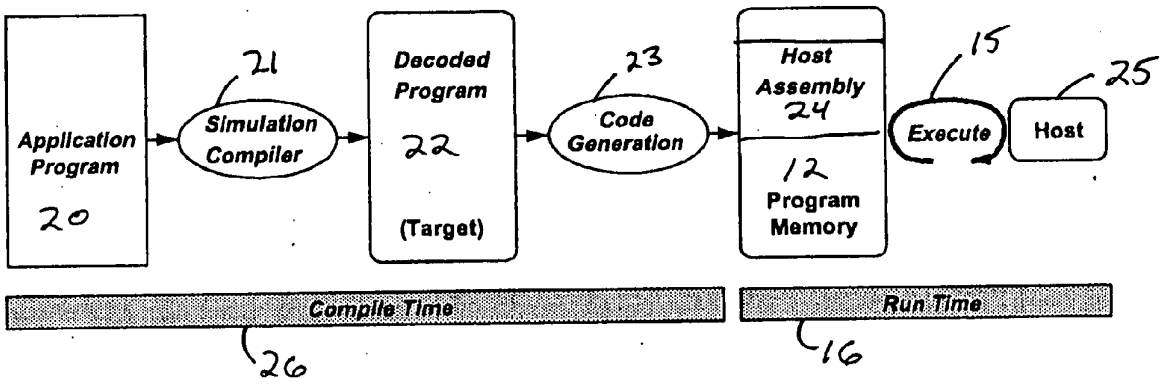


FIG. 2

PRIOR ART

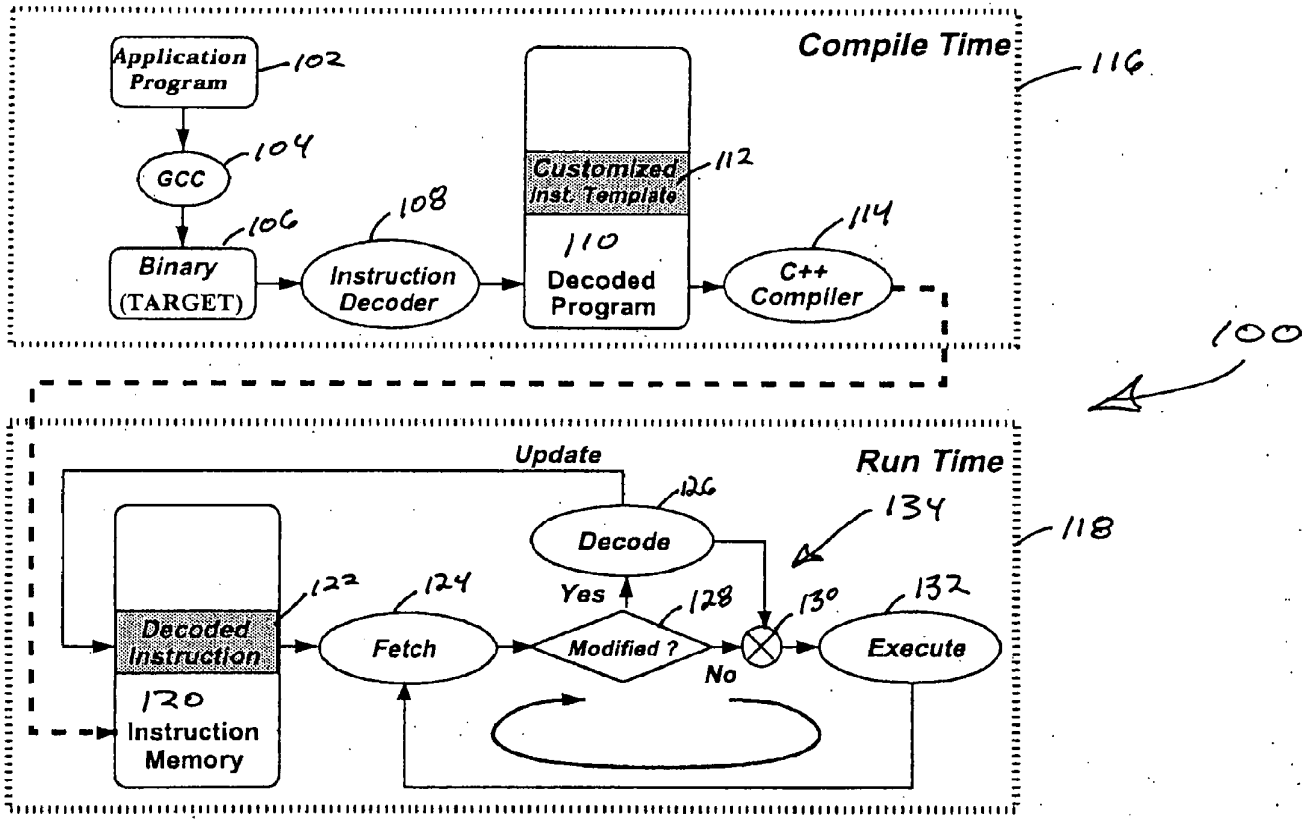


FIG. 3